Truck Platooning: week 7

Changyao Zhou, Jiaxin Pan

February 8, 2022

1 Introduction of the project

The goal of this project is to chase a chosen target vehicle by predicting the steering angle and throttle input of the ego-vehicle, given the velocity of both vehicles and images taken from cameras.

The control values (steering angle and throttle) computed by Model Predictive Control (MPC) are used as ground-truth. MPC is an advanced method of process control that is used to control a process while satisfying a set of constraints[WIK]. It takes a prediction horizon instead of a single time step into account, and aims to get an optimal control result minimizing the objective function within the prediction horizon. For learning the MPC controller, we take [MPC] as reference.

The project is designed to be realized in the following steps:

1). Control the ego-vehicle with Model Predictive Control (MPC) in the Carla simulator

2). Given x, y-location, yaw angle and velocity of both vehicles, train a fully connected neural network to control ego-vehicle by using the MPC output as ground-truth.

2). Given object velocity, ego-velocity and RGB image, train a neural network to control the ego-vehicle by using the MPC output as ground-truth

3). Further improvement by changing the input of the neural network

2 Overview of this week

Last week we tried to train a MLP model based on delta inputs. The loss was however fairly high and we could not get a desirable result. Then we switched to inputs with Δx, Δy and two yaw angles, which outputs a much better result.This week we further evaluate the model with 2-yaw inputs on unseen trajectories, testing if the model generalizes well. What's more, we tried to transform the input states of the target vehicle-/ego-vehicle and compute their relative pose, which are used as inputs for the MLP model. The model is then tested on unseen trajectories, which shows that this prepossessing step is really powerful.

What's more, we tried to train the CNN model once again on simple architecture and compared its output to the output from pretrained Alexnet.

All results of models mentioned above were tested with the 'online' mode in CARLA, where the target vehicle is set in 'autopilot' mode. All the videos are stored under https://syncandshare.lrz.de/getlink/fi5mLXJFkjPaoKYwDLkAVeEE/week7.

3 Training of MLP Model

3.1 Evaluation with 2-yaw inputs

Last week we trained a model with Δx, Δy and two yaw angles. This week we tried to test the model on unseen trajectories of the same town and in another town .

All the videos are stored under https://syncandshare.lrz.de/getlink/fiXEPir2BpQ9nRfzWTUH8mQC/2-yaw%20inputs.

As can be seen from the videos, the model seems to work well with sharp turns in another town. In the video 'town03_2.mp4', there is even a round turn in another town, which requires very precise steering control values, and the ego vehicle can follow the target vehicle in a reasonable way.

3.2 Training with Δv inputs

Another option of the inputs is to use the Δx, Δy, Δv and two yaw angles. The problem here is that, the throttle values, which let the target vehicle drives at a constant speed, are different with respect to different gears. From the data we collected, it can be seen that a throttle around $0.34m/s^2$ is needed to maintain a constant speed of 5m/s and the throttle for a constant speed of 10 m/s is $0.5m/s^2$.

Therefore, if we replace the 2 velocities with Δv, the ground truth throttle values maybe somehow contradictory when Δv is 0. Thus we didn't train a model with the Δv inputs.

3.3 Training with transformed inputs

3.3.1 Data Prepossession

Besides the delta inputs for the MLP Model, we tried to use the relative poses between two vehicles as input, which means that we transform the positions of the two vehicles into the same coordinate and then compute their relative rotation and translation, which are used as inputs.

To compute the relative pose, first we should compute the transformation matrix from world to the two vehicles respectively.

$$Tr = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \tag{1}$$

$$= \begin{bmatrix} cos(\theta) & -sin(\theta) & x \\ sin(\theta) & cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where the x, y and θ are the position and yaw angle of the vehicle.

Then the transformation matrix from one vehicle to another can be computed with:

$$Tr12 = \begin{bmatrix} R_1 & T_1 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} R_2 & T_2 \\ 0 & 1 \end{bmatrix} \tag{3}$$

$$= \begin{bmatrix} R_1^T & -R_1^T T_1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_2 & T_2 \\ 0 & 1 \end{bmatrix} \tag{4}$$

$$= \begin{bmatrix} R_1^T R_2 & R_1^T(T_2 - T_1) \\ 0 & 1 \end{bmatrix} \tag{5}$$

$$= \begin{bmatrix} \Delta R & \Delta T \\ 0 & 1 \end{bmatrix} \tag{6}$$

where ΔR and ΔT are used as inputs of the MLP Model.

3.3.2 Training Result

After training for around 350 epochs, we got a validation loss of 1.7e-3 for throttle and 4e-3 for steering angle, which is almost 10 times lower in comparison to the mean validation loss of 3e-2 we got last week with $\Delta\theta$.

We tested the model on unseen trajectories in the same town in the other towns, as well as with 2 ego vehicles. All the videos are stored under `https://syncandshare.lrz.de/getlink/fiXBVcTgUMYnVe9buo8pYXBt/transformed%20inputs`.

Though there some unexpected trajectories are included in these tests, such as sharp turns and sudden stop (in 'town04_unseen_2fol' and in 'town05_unseen_2fol.mp4'), the model can always output reasonable control values.

As can be concluded from the results, the model is very stable and also generalizes very well.

4 Training of CNN Model

4.1 Simple Architecture

We first tried to use a simple model, which is very similar from the model in this paper [KWC21]. The architecture is shown in Table 1.

Table 1: CNN Architecture

Layer Number	Layer Type	Layer Input	Layer Output
1	Convolution	3x128x128	30x62x62
2	MaxPooling	30x62x62	30x31x31
3	ReLU activation	30x31x31	30x31x31
4	Convolution	30x31x31	30x14x14
5	MaxPooling	30x14x14	30x7x7
6	ReLU activation	30x7x7	30x7x7
7	Convolution	30x7x7	30x2x2
8	MaxPooling	30x2x2	30x1x1
9	Flatten	32x1x1	32
10	Linear Layer	32	64
11	ReLU activation	64	64
12	Linear Layer	64	32
13	ReLU activation	32	32
14	Linear Layer	32	2

After training for around 100 epochs, the training loss is around 1e-2 and validation loss is around 2e-2. Since the training loss is not low enough, this may be an underfitting problem, which means that the model is too simple.

4.2 Model with pretrained Alexnet

Since there is an underfitting problem with the simple model, we switched to the pretrained Alexnet, which has a more complicated architecture and works well for our first trail in week 5.

The model is trained for around 50 epochs and the validation loss was around 6.5e-4 for both control values, while the validation loss for both were around 1.8e-2, which is a little bit lower than the simple model but can still be considered as not good enough in comparison to the MLP model. Since the training loss is already extremely low, there might be an overfitting problem with this model.

We tested the model on the seen and unseen trajectories. All the videos are stored under `https://syncandshare.lrz.de/getlink/fiHM3cRBsALtgqZmKnNRVP6C/CNN%20Model`.

As can be seen from the videos, and model output is not as stable as the MLP output and not generalizes that well. For the trajectory in the video 'seen_2', we have collected different routes for the intersection, which might be the reason that the model gets confused at the intersection.

4.3 Image Prepossession

4.3.1 Image normalization

We normalized the pixels in the images to a range of [-1, 1] with:

$$torchvision.transforms.Normalize([0.485, 0456, 0.406], [0.229, 0.224, 0.225]) \tag{7}$$

It turns out that this has a large impact on the training. We tested this with the simple model, whose validation loss is around 9e-2 without normalization, and the validation loss reached 2e-2 with normalization.

4.3.2 Conversion to Grayscale Images

Since the color information in the images actually does not have a great impact on the control value and they may make the images more noisy, we changed the images to grayscale images. We tested with the pretrained Alexnet and it turns out that there is no large difference between the loss of the model trained with grayscale and RGB images.

5 Conclusion and future work

In this week, we first tried to train the MLP with relative pose as inputs and get a extremely low loss. We tested the model with unseen trajectory even in other towns and with 2 ego vehicles, which all worked quite well.

For the CNN, we tested with the simple architecture as well as the pretrained Alexnet, both did not output a perfect result. In the next week we will try with other architectures or prepossessing methods for the CNN Model.

References

[KWC21] Qadeer Khan, Patrick Wenzel, and Daniel Cremers. Self-supervised steering angle prediction for vehicle control using visual odometry. In *International Conference on Artificial Intelligence and Statistics*, pages 3781–3789. PMLR, 2021.

[MPC] Model Predictive Control. https://github.com/WuStangDan/mpc-course-assignments.

[WIK] Model Predictive Control. https://en.wikipedia.org/wiki/Model_predictive_control.